DW_lp_piped_prod_sum

Low Power Pipelined Sum of Products

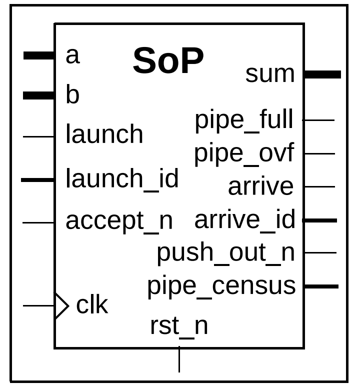
Version, STAR, and myDesignWare Subscriptions: IP Directory

DesignWare minPower Component

Features and Benefits

Revision History

- Built-in pipelining and power management
- Automatically enables Design Compiler to retime registers
- Operand isolation capability on `a` and `b` inputs
- Parameterized operand widths
- Parameterized pipeline stages
- Launch identifier tracking propagation
- Operand Isolation capability on `a` and `b`



Description

DW_lp_piped_prod_sum performs a pipelined summation of a set of products (`a(i) × b(i)`) with the added benefit of power savings. Pipeline control is integrated and applied to pipelined register levels (when configured in) to minimize power consumption. Pipeline register re-timing is automatically enabled for balancing between logic stages.

Table 1-1 Pin Descriptions

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Asynchronous or synchronous reset depending on rst_mode parameter (active low)
a	*a_width* bits	Input	Concatenated multiplier
b	*b_width* bits	Input	Concatenated multiplicand
sum	*sum_width* bits	Output	Sum of products
launch	1 bit	Input	Control to begin a new multiply operation
launch_id	*id_width* bits	Input	Identifier for the corresponding asserted `launch`
pipe_full	1 bit	Output	Upstream notification that pipeline is full
pipe_ovf	1 bit	Output	Status Flag indicating pipe overflow
accept_n	1 bit	Input	`sum` result accepted from downstream logic (active low)
arrive	1 bit	Output	`sum` result is valid

Table 1-1 Pin Descriptions (Continued)

Pin Name	Width	Direction	Function
arrive_id	*id_width* bits	Output	`launch_id` from the originating `launch` that produced the `sum` result
push_out_n	1 bit	Output	Push performed to downstream FIFO element (active low)
pipe_census	M bits	Output	Number of pipeline register levels currently occupied Note: The value of M is equal to the larger of '1' or ceil(log2(in_reg+stages+out_reg)) Example: if in_reg = 1, stages = 2, out_reg = 1, then M = 2

Table 1-2 Parameter Description

Parameter	Values	Description
a_width	≥ 1 Default: 8	Word length of `a`
b_width	≥ 1 Default: 8	Word length of `b`
num_inputs	≥ 1 Default: 2	Number of inputs
sum_width	≥ 1 Default: 17	Word width of `sum`
id_width	1 to 1024 Default: 8	Width of `launch_id`
in_reg	0 or 1 Default: 0	Input register control ■ 0 = No input register ■ 1 = Include input register
stages	1 to 1022 Default: 4	Number of pipeline stages
out_reg	0 or 1 Default: 0	Output register control ■ 0 = No output register ■ 1 = Include output register
tc_mode	0 or 1 Default: 0	Two's complement control ■ 0 = Unsigned ■ 1 = Signed
rst_mode	0 to 1 Default: 0	Reset mode ■ 0 = Asynchronous reset ■ 1 = Synchronous reset

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
op_iso_mode	0 to 4 Default: 0	Operand isolation mode (controls datapath gating for minPower flow) Allows you to set the style of minPower datapath gating for this module ■ 0 = Use the DW_lp_op_iso_mode[a] synthesis variable ■ 1 = 'none' ■ 2 = 'and' ■ 3 = 'or' ■ 4 = Preferred gating style: ‘and’ Datapath gating is inserted only when there are no input registers on the operands at the component boundary. When inserted, datapath gating circuits are placed immediately after the input ports of the component (see Figure 1-2 on page 5).

a. The DW_lp_op_iso_mode synthesis variable is available only in Design Compiler. DW_lp_op_iso_mode sets a global style of datapath gating. To use the global style, set *op_iso_mode* to '0', Note that If the *op_iso_mode* parameter is set to '0' and DW_lp_op_iso_mode is either not set or set to 0', then no datapath gating is inserted for this component.

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	■ DesignWare (P-2019.03 and later) ■ DesignWare-LP[a] (before P-2019.03)

a. For Design Compiler versions before P-2019.03, see “Enabling minPower” on page 14.

Table 1-4 Simulation Models

Model	Function
DW03.DW_LP_PIPED_PROD_SUM_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_lp_piped_prod_sum_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_lp_piped_prod_sum.v	Verilog simulation model source code

Block Diagram

Figure 1-1 shows the block diagram of the DW_lp_piped_prod_sum component:

Figure 1-1 DW_lp_piped_prod_sum Basic Block Diagram

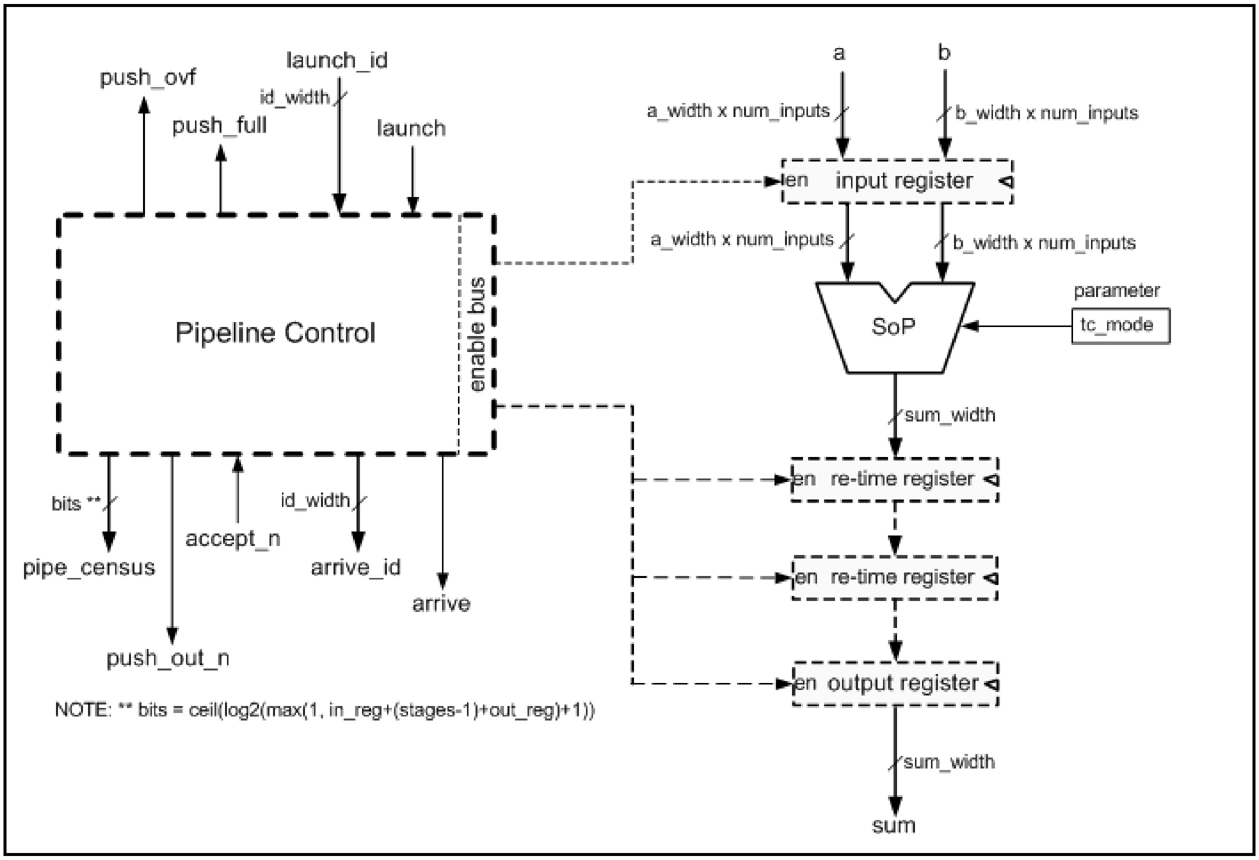
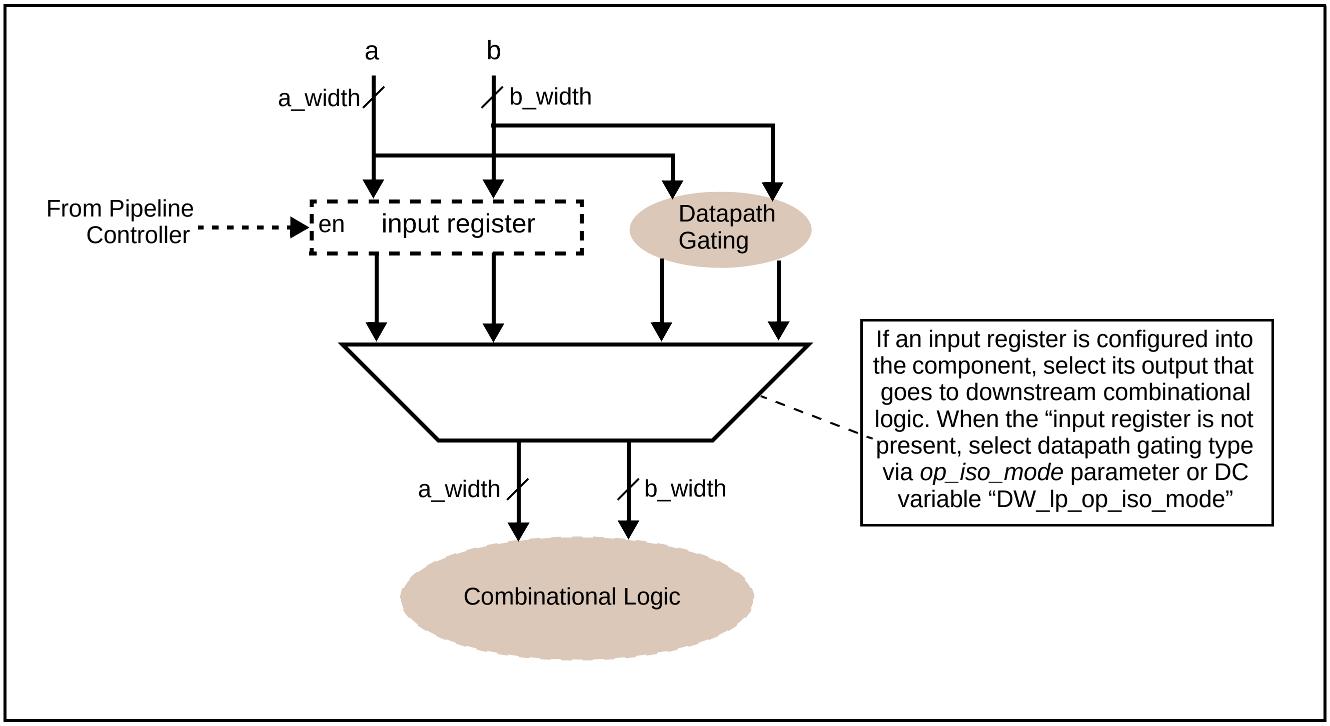


Figure 1-2 shows where datapath gating is inserted when the *op_iso_mode* parameter enables it.

Figure 1-2 Location of Datapath Gating (If Inserted)



Functional Description

The equation for 'sum' is:

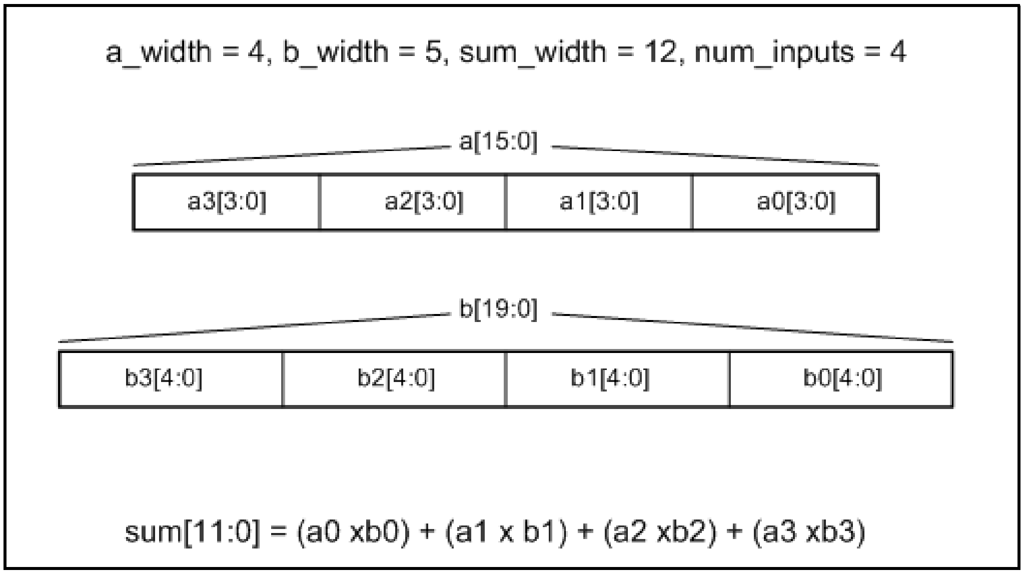
$$sum(k-1:0) = \sum_{j=0}^{N-1} a[(j+1) \times m-1:j \times m] \times b[(j+1) \times n-1:j \times n]$$

where:
m = *a_width*
n = *b_width*
N = *num_inputs*
k = *sum_width*

The set of coefficients to be multiplied and summed must be concatenated into two vectors, each with a width of *num_inputs* × *a_width* and *num_inputs* × *b_width*. These two vectors are connected to the `a` and `b` pins.

Internally, DW_lp_piped_prod_sum disassembles the individual words from `a` and `b` and then performs the sum of products. The *tc_mode* parameter determines whether the input and output data is interpreted as unsigned numbers (*tc_mode* = 0) or signed (*tc_mode* = 1).

Figure 1-3 Functional Operation



Pipelining

The DW_lp_piped_prod_sum is configurable embed pipeline register levels. Setting the value for the parameters *in_reg*, *stages*, and *out_reg* (see Table 1-5 below) determines the number of pipeline register levels that are inserted. Therefore, depending on the parameter *in_reg*, *stages*, and *out_reg* settings, the number of clock cycles for the `sum` result to propagate varies.

This DW_lp_piped_prod_sum is designed to make it easy to pipeline the multiplier using the register retiming features of Design Compiler (DC). It also contains parameter controlled input and output registers which will stay in place at their respective block boundary - they are not allowed to be moved by DC register retiming features. The input and output registers are not available when using DC versions earlier than A-2007.12.

The parameter *stages* refers to the number of logic stages desired after register retiming is performed. The number of register levels is not necessarily the same as the number of logic stages. If no input or output registers are used (*in_reg* = 0 or *out_reg* = 0), then there is one fewer register level than logic *stages*. If either an input register or output register is specified, then the number of register levels is the same as the number of logic *stages*. If both input and output registers are specified, then the number of register levels is the number of logic *stages* + 1. This is described in Table 1-5. The number of pipeline register levels that can be retimed is always *stages* - 1.

Table 1-5 Number of Pipeline Register Levels

in_reg	out_reg	Number of Pipeline Register Levels
0	0	stages - 1
0	1	stages
1	0	stages
1	1	stages + 1

Pipeline Control and Power Savings

Running in parallel to the pipeline register levels is pipeline control logic (as seen in Figure 1-1 on page 4) that monitors the activity. In cases where there is inactivity on a particular register level of the pipeline, the pipeline control disables those levels to promote power savings. Furthermore, if using the Synopsys Power Compiler tool, the presence of the pipeline control and its wiring to the pipeline register levels provides an opportunity for increased power reduction in the form of clock gating.

Along with the potential power savings that the pipeline control provides, it can be utilized to improve performance in cases where intermittent launch operations are present and there contains first-in first-out (FIFO) structures upstream and downstream of the DW_lp_piped_prod_sum. The handshake is made between the DW_lp_piped_prod_sum and the external FIFOs via the `accept_n` and `pipe_full` ports. Effectively, the DW_lp_piped_prod_sum can be considered part of the external FIFO structures. The performance gain comes when inactive (bubbles) stages are detected. These pipeline 'bubbles' are removed to produce a contiguous set of active pipeline stages. The result is empty pipeline slots at the head of (or entering) the DW_lp_piped_prod_sum pipeline for new operations to be launched. Advancing the shifting of operations through the pipeline when a valid product result is available (`arrive = 1`) is controlled by the `accept_n` input. When the multiplier pipeline is full of active entries, the `pipe_full` output is 1. To disable this feature in cases where no external FIFOs are present, set the `accept_n` input to 0 which will effectively eliminate any flow control. At the same time, the `pipe_full` output would always be 0.

To assist in tracking of launched operands, the pipeline control logic provides interface ports called `launch_id` and `arrive_id`. The `launch_id` input is assigned a value during an active launch operation. Given that `launch_id` values are unique in successive launch operations, the product results can be distinguished from one another with the assertion of `arrive` and the associated `arrive_id`. The `arrive_id` is the `launch_id` from the originating `launch` that produced the valid product result.

No Pipeline Register Levels Specified

In cases where no pipelining is required through the DW_lp_piped_prod_sum (*in_reg* = 0, *stages* = 1, and *out_reg* = 0), the pipeline control flow control handshaking/status signals still remain active and meaningful with one exception. The `pipe_census`, which is intended to count the number of active pipeline register levels, becomes irrelevant and is fixed to 0. For timing waveforms, wee Figure 1-7 on page 11.

Also, under this configuration simulation models will not drive X's on `sum` output when `arrive` (and `launch`) are not asserted. Note in this configuration, `arrive = launch`. So, only when `launch` and `arrive` are asserted is meaningful `sum` results driven.

Timing Waveforms

Figure 1-4 shows a case where there are two pipeline register levels since *in_reg is* = 0, *stages* = 2, and *out_reg* = 1. Launching is performed while `accept_n` is de-asserted causing the pipeline to fill up. This is indicated by `pipe_full` going to 1 while `accept_n` is 1. The `pipe_census[1:0]` value is 2 which indicates that all the pipeline register levels contain active results. Notice that the first `sum[8:0]` result of -32 is available as indicated by `arrive` being 1. Also, the identification tracking from the `launch_id[3:0]` of 1 is seen upon arrival with the matching value on `arrive_id[3:0]`.

At the point that the pipeline is full, `accept_n` is asserted (0) to begin emptying the pipeline. Note that `pipe_full` de-asserts when `accept_n` is asserted, but the `pipe_census[1:0]` value still indicates 2 the next clock cycle since a launch coincided with the asserted `accept_n`. Once the launching activity ceases, the continued assertion of `accept_n` drains the pipeline of active `sum[15:0]` results with `pipe_census[1:0]` eventually going to 0. The DW_lp_piped_prod_sum is configured to operate in two's complement (*tc_mode* = 1).

NOTE: `sum[8:0]` is displayed in signed decimal format, but `a[7:0]` and `b[7:0]` are displayed in hexadecimal format.

Figure 1-4 Launching Until Full, Accepting Until Empty

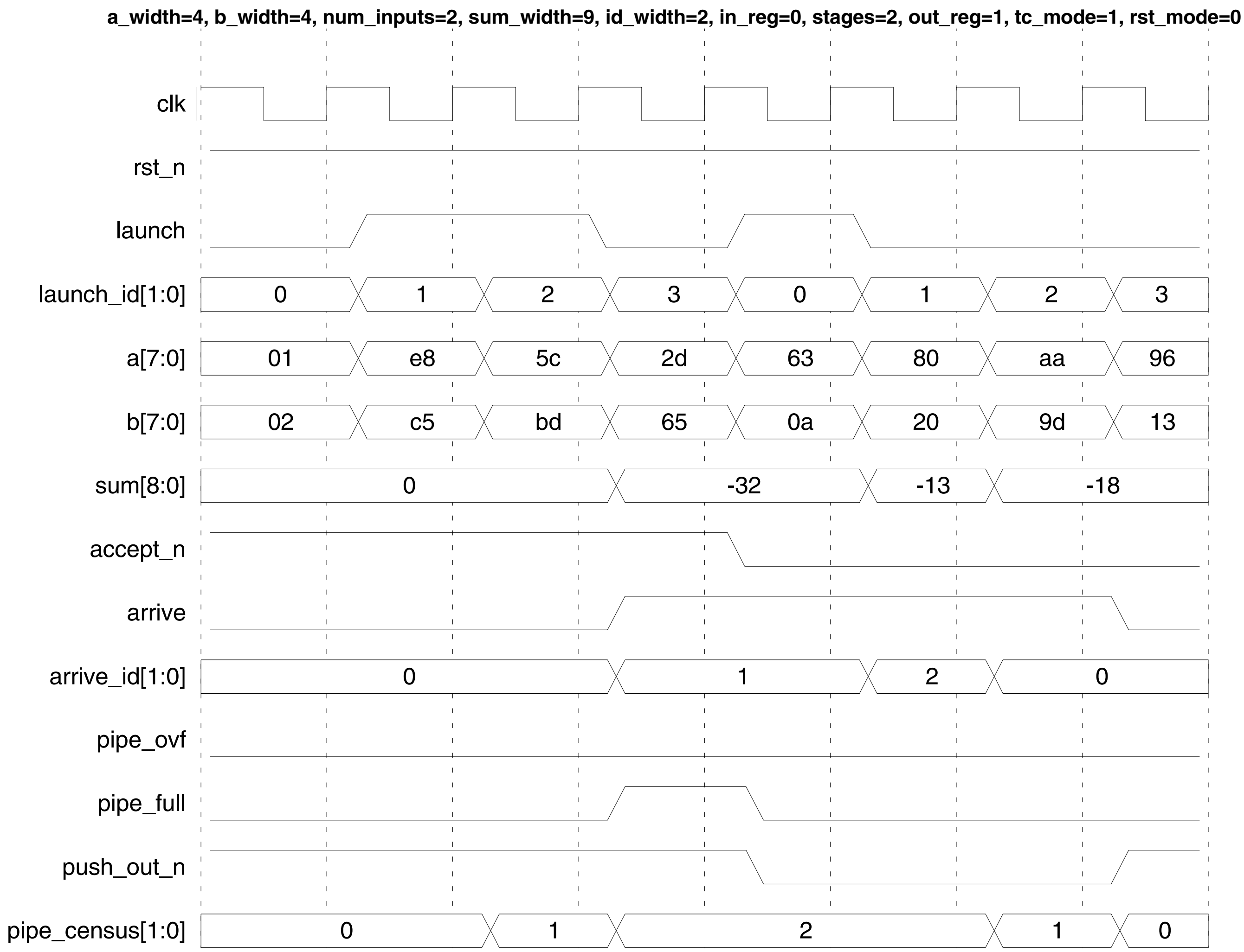
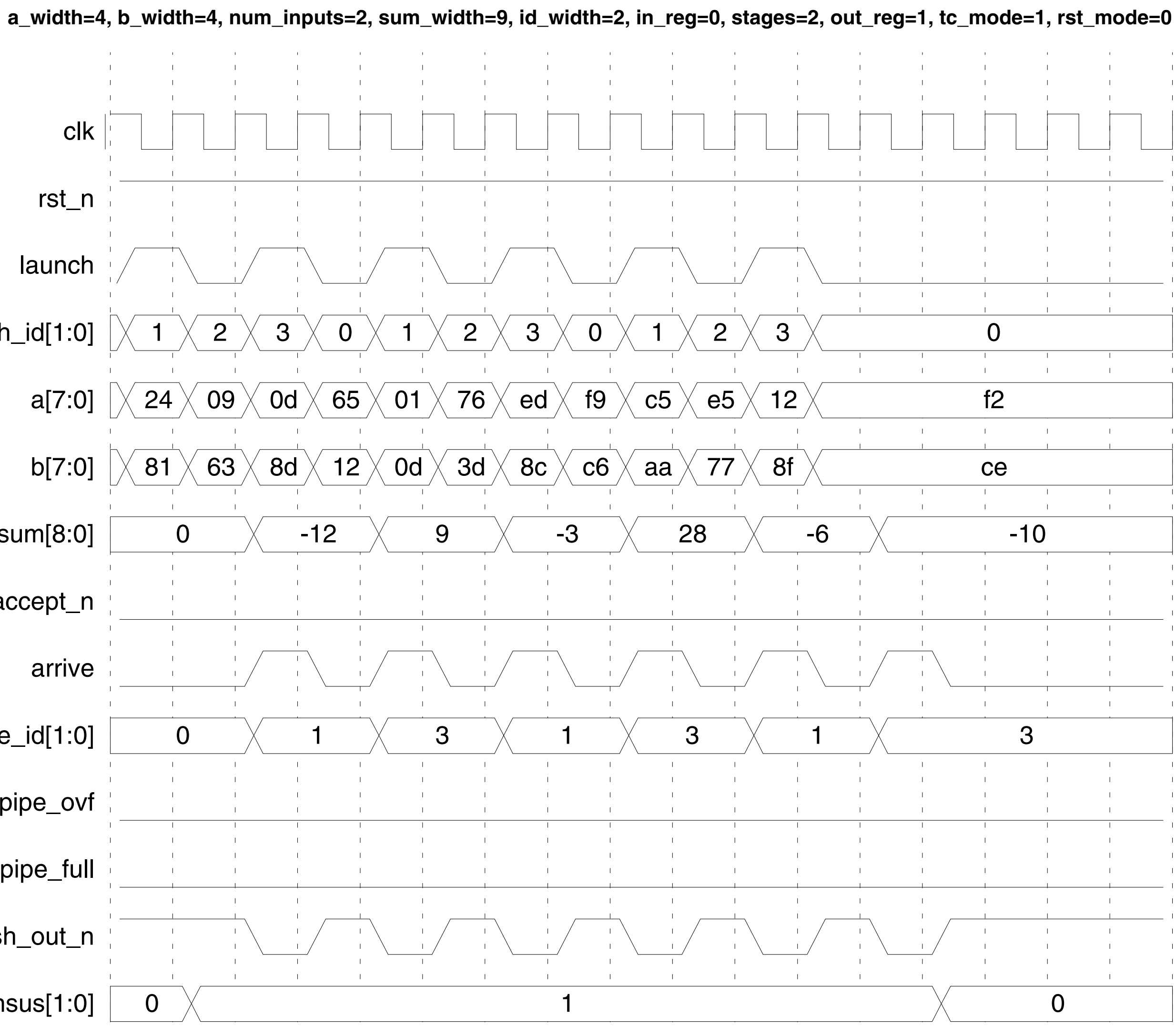


Figure 1-5 shows a case where `launch` is asserted every other clock cycle while `accept_n` is always asserted (0). There are 4 pipeline register levels. So, the first `sum` result of '-12' (`arrive_id[2:0]` of 1) arrives after the fourth rising-edge of `clk` from the first asserted `launch` with accompanying `launch_id[2:0]` of 1. Any values of `a[7:0]` and `b[7:0]` are ignored when `launch` is 0.

The DW_lp_piped_prod_sum is configured to operate in unsigned mode (*tc_mode* = 1).

NOTE: `sum[8:0]` is displayed in signed decimal format, but `a[7:0]` and `b[7:0]` are displayed in hexadecimal format.

Figure 1-5 Launch Every Other Cycle with Asserted accept_n



The DW_lp_piped_prod_sum is configured to operate in unsigned mode (*tc_mode=0*).

NOTE: `sum[13:0]` is displayed in unsigned decimal format, but `a[23:0]` and `b[11:0]` are displayed in hexadecimal format.

Figure 1-6 depicts a pipeline overflow condition. This is the same configuration as shown in Figure 1-5 on page 9. The `pipe_ovf` output is registered and gets asserted following the rising-edge of `clk` when the pipeline is full (`pipe_full` is 1), `launch` is asserted (1), and `accept_n` is not asserted (1). In this situation, the launched operation is ignored and the pipeline contents are preserved.

Figure 1-6 Pipeline Overflow

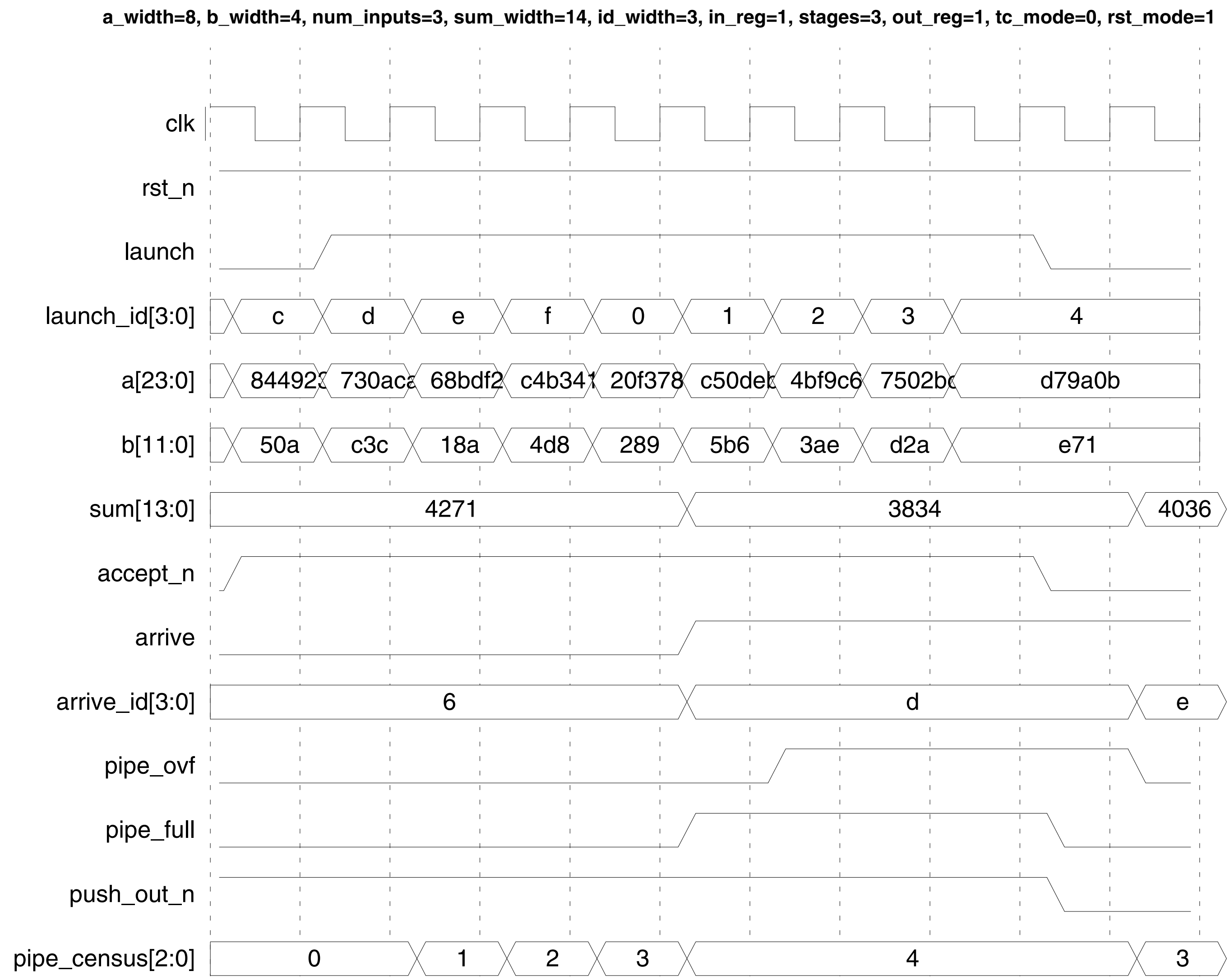


Figure 1-7 depicts a scenario when there is no pipelining configured into the DW_lp_piped_prod_sum which is defined when *in_reg* = 0, *stages* = 1, and *out_reg* = 0. Thus, the product result is a pure combinational logic path from `a[7:0]` and `b[7:0]`. The flow control/status outputs `arrive`, `arrive_id[7:0]`, `pipe_full`, `pipe_ovf`, `push_out_n` still have meaning. However, the output `pipe_census` has no meaning since no pipeline register levels exist. Hence, `pipe_census` will always be driven to 0.

Notice that when `launch` is asserted and `accept_n` is not, the register output `pipe_ovf` goes to 1. This is due to the fact that when `accept_n` is 1, it implies that the downstream device cannot accept any more results. Thus, a launch under this condition will result in overrun and the subsequent product result is lost.

The DW_lp_piped_prod_sum is configured to operate in unsigned mode (*tc_mode* = 0).

NOTE: `sum[13:0]` is displayed in unsigned decimal format, but `a[23:0]` and `b[11:0]` are displayed in hexadecimal format.

Figure 1-7 No Pipeline Specified (*in_reg* = 0, *stages* = 1, *out_reg* = 0, *tc_mode* = 0)

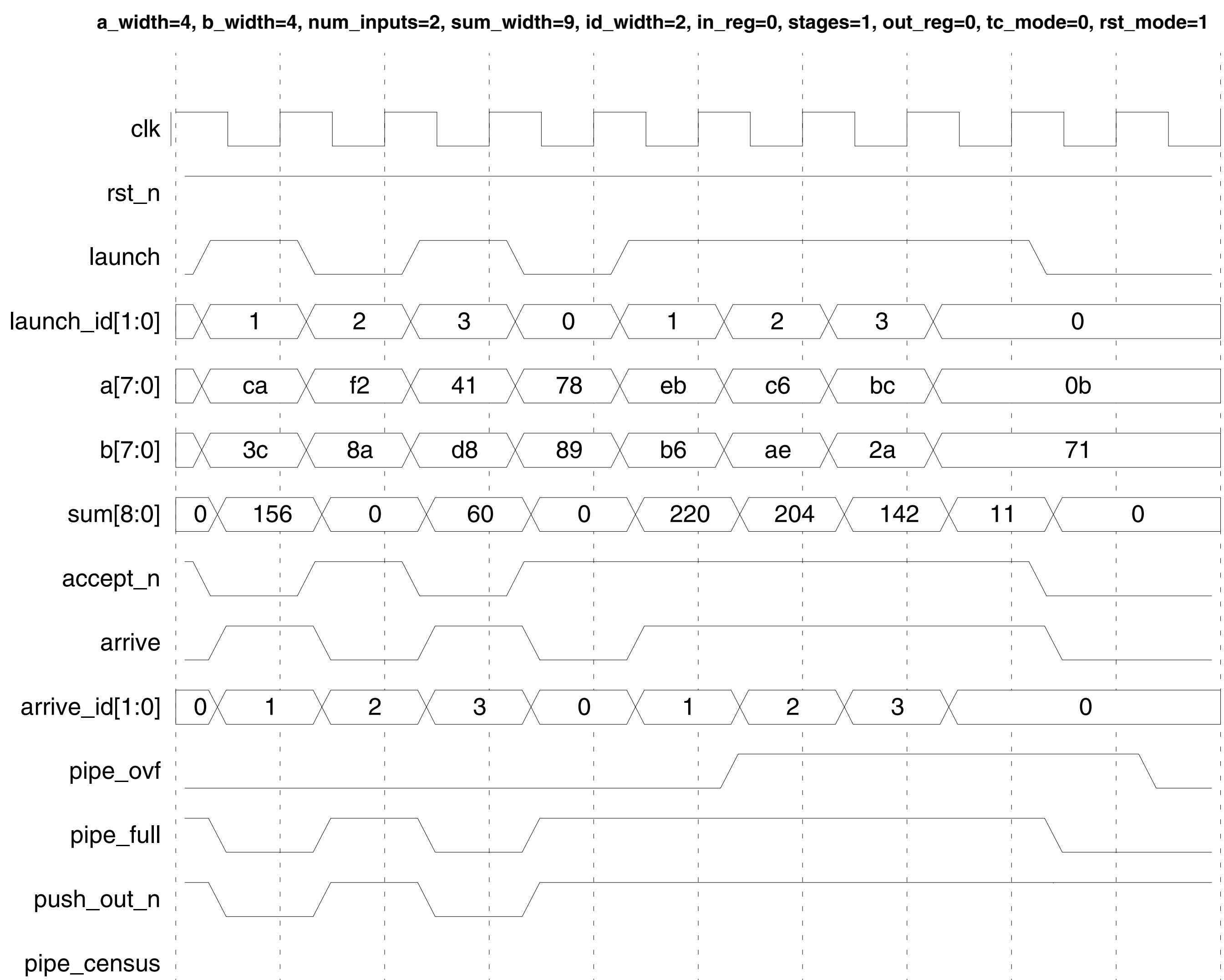


Figure 1-8 shows the affects that the assertion of `rst_n` while configured for asynchronous resetting (*rst_mode* = 0).

Figure 1-8 Asynchronous Reset Behavior (*rst_mode* = 0)

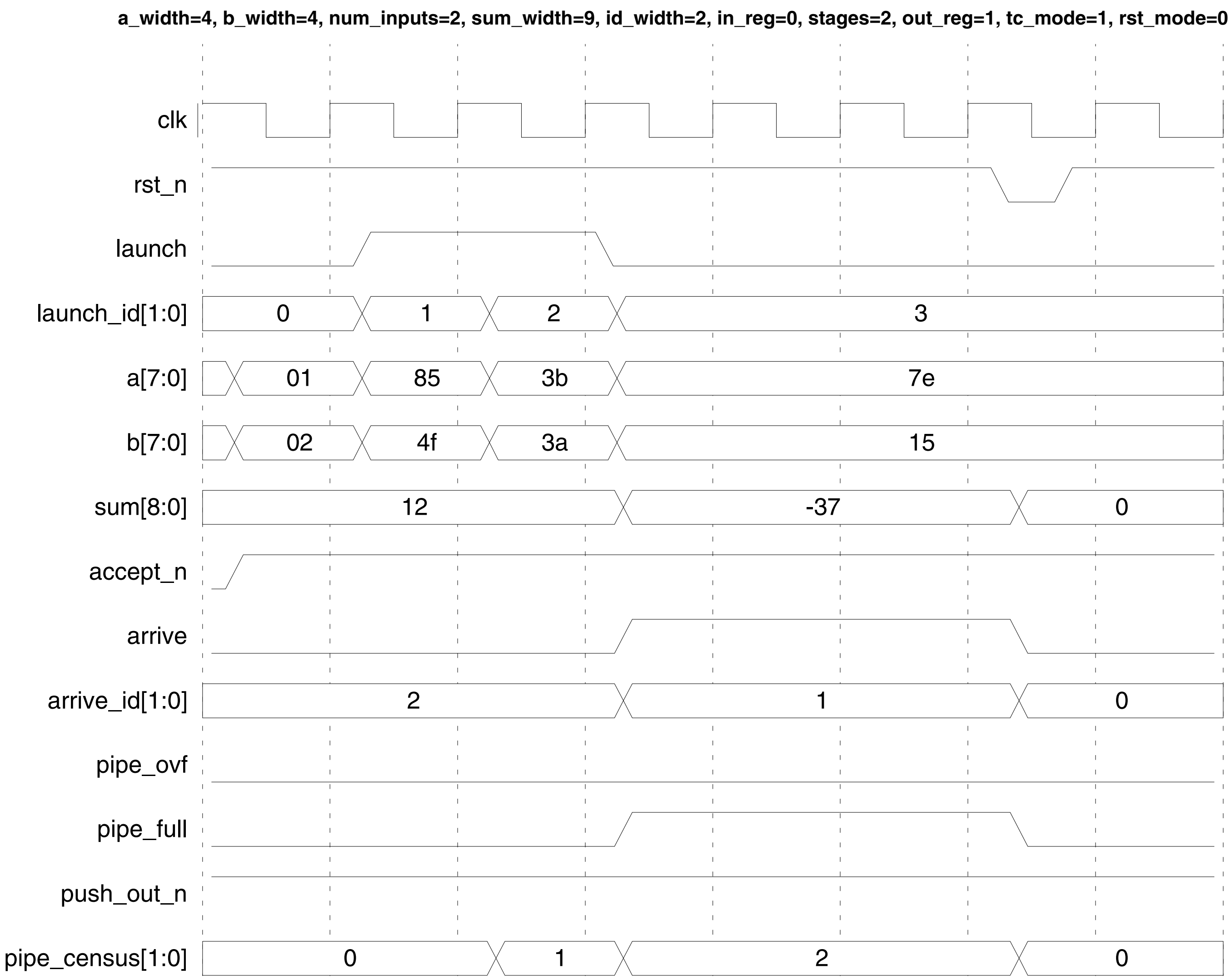
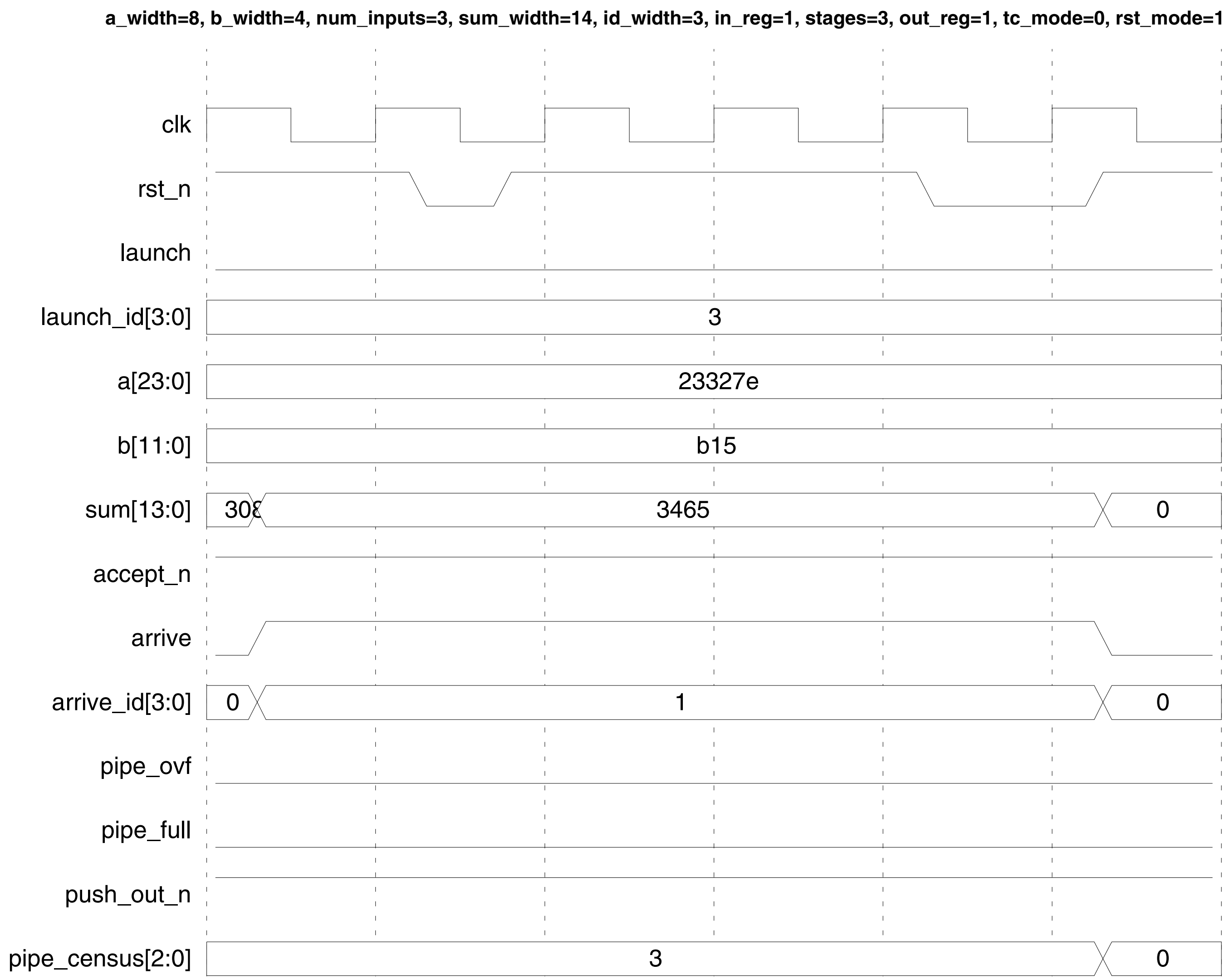


Figure 1-9 shows the affects of asserting `rst_n` while configured for synchronous resetting (*rst_mode* = 1). Only when a 0 state of `rst_n` is sampled by the rising-edge of `clk` does the clearing of register elements occur.

Figure 1-9 Synchronous Reset Behavior (*rst_mode* = 1)



Enabling minPower

In Design Compiler (version P-2019.03 and later) and Fusion Compiler, you can instantiate this component and use all its features without special settings.

For versions of Design Compiler before P-2019.03, enable minPower as follows:

```
set synthetic_library {dw_foundation.sldb dw_minpower.sldb}
set link_library {* $target_library $synthetic_library}
```

Related Topics

- Math – Arithmetic Overview
- DesignWare Building Blocks User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_Foundation_comp.all;

entity DW_lp_piped_prod_sum_inst is
      generic (
        inst_a_width : POSITIVE := 8;
        inst_b_width : POSITIVE := 8;
            inst_num_inputs : POSITIVE  := 2;
            inst_sum_width : POSITIVE  := 17;
        inst_id_width : POSITIVE := 16;
        inst_in_reg : NATURAL := 0;
        inst_stages : POSITIVE := 3;
        inst_out_reg : NATURAL := 0;
        inst_tc_mode : NATURAL := 0;
        inst_rst_mode : NATURAL := 0;
        inst_op_iso_mode : NATURAL := 0
        );
      port (
        inst_clk : in std_logic;
        inst_rst_n : in std_logic;
        inst_a : in std_logic_vector((inst_a_width*inst_num_inputs)-1 downto 0);
        inst_b : in std_logic_vector((inst_b_width*inst_num_inputs)-1 downto 0);
        sum_inst : out std_logic_vector(inst_sum_width-1 downto 0);
        inst_launch : in std_logic;
        inst_launch_id : in std_logic_vector(inst_id_width-1 downto 0);
        pipe_full_inst : out std_logic;
        pipe_ovf_inst : out std_logic;
        inst_accept_n : in std_logic;
        arrive_inst : out std_logic;
        arrive_id_inst : out std_logic_vector(inst_id_width-1 downto 0);
        push_out_n_inst : out std_logic;
        pipe_census_inst : out std_logic_vector(1 downto 0)
        );
    end DW_lp_piped_prod_sum_inst;


architecture inst of DW_lp_piped_prod_sum_inst is

begin

    -- Instance of DW_lp_piped_prod_sum
    U1 : DW_lp_piped_prod_sum
    generic map  ( a_width => inst_a_width,
                   b_width => inst_b_width,
                       num_inputs => inst_num_inputs,
```

```
                    sum_width => inst_sum_width,
                id_width => inst_id_width,
                in_reg => inst_in_reg,
                stages => inst_stages,
                out_reg => inst_out_reg,
                tc_mode => inst_tc_mode,
                rst_mode => inst_rst_mode,
                op_iso_mode => inst_op_iso_mode )

    port map ( clk => inst_clk,
                rst_n => inst_rst_n,
                a => inst_a,
                b => inst_b,
                sum => sum_inst,
                launch => inst_launch,
                launch_id => inst_launch_id,
                pipe_full => pipe_full_inst,
                pipe_ovf => pipe_ovf_inst,
                accept_n => inst_accept_n,
                arrive => arrive_inst,
                arrive_id => arrive_id_inst,
                push_out_n => push_out_n_inst,
                pipe_census => pipe_census_inst );

end inst;

-- Configuration for use with a VHDL simulator
-- pragma translate_off
library DW03;
configuration DW_lp_piped_prod_sum_inst_cfg_inst of DW_lp_piped_prod_sum_inst is
  for inst
  end for; -- inst
end DW_lp_piped_prod_sum_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_lp_piped_prod_sum_inst( inst_clk, inst_rst_n, inst_a, inst_b, sum_inst,
inst_launch, inst_launch_id,
          inst_accept_n, arrive_inst, arrive_id_inst, pipe_full_inst,
          pipe_ovf_inst, push_out_n_inst, pipe_census_inst );

parameter a_width = 8;
parameter b_width = 8;
parameter num_inputs = 2;
parameter sum_width = 17;
parameter id_width = 8;
parameter in_reg = 0;
parameter stages = 3;
parameter out_reg = 0;
parameter tc_mode = 0;
parameter rst_mode = 0;
parameter op_iso_mode = 0;


`define census_width 2  // ceil(log2(max(1, in_reg+(stages-1)+out_reg)+1))

input inst_clk;
input inst_rst_n;
input [(a_width*num_inputs)-1:0] inst_a;
input [(b_width*num_inputs)-1:0] inst_b;
output [sum_width-1:0] sum_inst;
input inst_launch;
input [id_width-1 : 0] inst_launch_id;
input inst_accept_n;
output arrive_inst;
output [id_width-1 : 0] arrive_id_inst;
output pipe_full_inst;
output pipe_ovf_inst;
output push_out_n_inst;
output [`census_width-1 : 0] pipe_census_inst;

    // Instance of DW_lp_piped_prod_sum
    DW_lp_piped_prod_sum #(a_width, b_width, num_inputs, sum_width, id_width, in_reg,
stages, out_reg, tc_mode, rst_mode, op_iso_mode)
      U1 ( .clk(inst_clk),
               .rst_n(inst_rst_n),
               .a(inst_a),
               .b(inst_b),
               .sum(sum_inst),
               .launch(inst_launch),
               .launch_id(inst_launch_id),
               .accept_n(inst_accept_n),
               .arrive(arrive_inst),
```

```
                  .arrive_id(arrive_id_inst),
                  .pipe_full(pipe_full_inst),
                  .pipe_ovf(pipe_ovf_inst),
                  .push_out_n(push_out_n_inst),
                  .pipe_census(pipe_census_inst)
                  );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2023	DWBB_202212.5	■ Updated version and date
March 2019	DWBB_201903.0	■ Clarified the op_iso_mode parameter in Table 1-2 on page 2 ■ Clarified licensing requirements in Table 1-3 on page 3 ■ Added Figure 1-2 on page 5 to clarify datapath gating ■ Added “Enabling minPower” on page 14 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Synopsys, Inc.
www.synopsys.com